Package 'polysat'

June 21, 2010

Version 0.1

Date 2010-06-01

Title Tools for Polyploid Microsatellite Analysis

Author Lindsay V. Clark <lvclark@ucdavis.edu>

Maintainer Lindsay V. Clark <lvclark@ucdavis.edu>

Depends combinat

Suggests ade4, adegenet

Description polysat is a collection of tools to handle microsatellite data of any ploidy (and samples of mixed ploidy) where allele copy number is not known in partially heterozygous genotypes. It can import and export data in ABI GeneMapper, Structure, ATetra, Tetrasat/Tetra, GenoDive, SPAGeDi, and binary presence/absence formats. It can calculate pairwise differences between individuals using a stepwise mutation model or infinite alleles model. It can assist the user in estimating the ploidy of samples, and lastly it can estimate allele frequencies in populations and calculate pairwise Fst values based on those frequencies.

License GPL (>=2)

URL http://openwetware.org/wiki/Polysat

R topics documented:

`Bruvo.distance` *Genetic Distance Metric of Bruvo et al*

Description

This function calculates the distance between two individuals at one microsatellite locus using the method of Bruvo et al. (2004).

Usage

```
Bruvo.distance(genotype1, genotype2, maxl=9, usatnt=2, missing=-9)
```

Arguments

`genotype1`	A vector of alleles for one individual at one locus. Allele length is in nucleotides or repeat count. Each unique allele corresponds to one element in the vector, and the vector is no longer than it needs to be to contain all unique alleles for this individual at this locus.
`genotype2`	A vector of alleles for another individual at the same locus.
`maxl`	If both individuals have more than this number of alleles at this locus, `NA` is returned instead of a numerical distance.
`usatnt`	Length of the repeat at this locus. For example `usatnt=2` for dinucleotide repeats, and `usatnt=3` for trinucleotide repeats. If the alleles in `genotype1` and `genotype2` are expressed in repeat count instead of nucleotides, set `usatnt=1`.
`missing`	A numerical value that, when in the first allele position, indicates missing data. `NA` is returned if this value is found in either genotype.

Details

Since allele copy number is frequently unknown in polyploid microsatellite data, Bruvo et al. developed a measure of genetic distance similar to band-sharing indices used with dominant data, but taking into account mutational distances between alleles. A matrix is created containing all differences in repeat count between the alleles of two individuals at one locus. These differences are then geometrically transformed to reflect the probabilities of mutation from one allele to another. The matrix is then searched to find the minimum sum if each allele from one individual is paired to one allele from the other individual. This sum is divided by the number of alleles per individual.

If one genotype has more alleles than the other, 'virtual alleles' must be created so that both genotypes are the same length. There are three options for the value of these virtual alleles, but `Bruvo.distance` only implements the simplest one, assuming that it is not known whether differences in ploidy arose from genome addition or genome loss. Virtual alleles are set to infinity, such that the geometric distance between any allele and a virtual allele is 1.

Value

A number ranging from 0 to 1, with 0 indicating identical genotypes, and 1 being a theoretical maximum distance if all alleles from `genotype1` differed by an infinite number of repeats from all alleles in `genotype2`. `NA` is returned if both genotypes have more than `maxl` alleles or if either genotype has the symbol for missing data as its first allele.

Note

The processing time is a function of the factorial of the number of alleles, since each possible combination of allele pairs must be evaluated. For genotypes with a sufficiently large number of alleles, it may be more efficient to estimate distances manually by creating the matrix in Excel and visually picking out the shortest distances between alleles. This is the purpose of the `maxl` argument. On my personal computer, if both genotypes had more than nine alleles, the calculation could take an hour or more, and so this is the default limit. In this case, `Bruvo.distance` returns `NA`.

Author(s)

Lindsay V. Clark

References

Bruvo, R., Michiels, N. K., D'Sousa, T. G., and Schulenberg, H. (2004) A simple method for calculation of microsatellite genotypes irrespective of ploidy level. *Molecular Ecology* **13**, 2101-2106.

See Also

`distance.matrix.1locus`, `meandistance.matrix`, `Lynch.distance`

Examples

```
Bruvo.distance(c(202,206,210,220),c(204,206,216,222))
Bruvo.distance(c(202,206,210,220),c(204,206,216,222),usatnt=4)
Bruvo.distance(c(202,206,210,220),c(204,206,222))
Bruvo.distance(c(202,206,210,220),c(204,206,216,222),maxl=3)
Bruvo.distance(c(202,206,210,220),c(-9))
```

`calcFst` *Calculate Wright's Pairwise FST*

Description

Given a data frame of allele frequencies and population sizes, `calcFst` calculates a matrix of pairwise Fst values.

Usage

```
calcFst(freqs, pops = row.names(freqs), loci = unique(as.matrix(
as.data.frame(strsplit(names(freqs), split = ".", fixed = TRUE),
stringsAsFactors = FALSE))[1, ]))
```

Arguments

freqs	A data frame of allele frequencies and population sizes such as that produced by estimate.freq. Each population is in one row, and a column called Genomes contains the relative size of each population. All other columns contain allele frequencies. The names of these columns are the locus name and allele name, separated by a period.
pops	A character vector. Populations to analyze, which should be a subset of row.names(freqs).
loci	A character vector indicating which loci to analyze. These should be a subset of the locus names as used in the column names of freqs.

Details

calcFst works by calculating HS and HT for each locus for each pair of populations, then averaging HS and HT across loci. FST is then calculated for each pair of populations as (HT-HS)/HT.

H values (expected heterozygosities for populations and combined populations) are calculated as one minus the sum of all squared allele frequencies at a locus. To calculte HT, allele frequencies between two populations are averaged before the calculation. To calculate HS, H values are averaged after the calculation. In both cases, the averages are weighted by the relative sizes of the two populations (as indicated by freqs$Genomes).

Value

A square matrix containing FST values. The rows and columns of the matrix are both named by population.

Author(s)

Lindsay V. Clark

References

Nei, M. (1973) Analysis of Gene Diversity in Subdivided Populations. *Proceedings of the National Academy of Sciences of the United States of America* **70**, 3321–3323.

See Also

estimate.freq

Examples

```
# create a data set (typically done by reading files)
mygenotypes <- array(list(-9), dim = c(6,2), dimnames =
list(paste("ind",1:6, sep=""),c("loc1","loc2")))
mygenotypes[,"loc1"] <- list(c(206), c(208,210), c(204,206,210),
c(196,198,202,208), c(196,200), c(198,200,202,204))
mygenotypes[,"loc2"] <- list(c(130,134), c(138,140), c(130,136,140),
c(138), c(136,140), c(130,132,136))
```

```
mypopinfo <- c(1,1,1,2,2,2)
names(mypopinfo) <- dimnames(mygenotypes)[[1]]

myploidies <- c(2,2,4,4,2,4)
names(myploidies) <- dimnames(mygenotypes)[[1]]

# calculate allele frequencies
myfreq <- estimate.freq(mygenotypes, popinfo = mypopinfo,
indploidies = myploidies)

# calculate pairwise FST
myfst <- calcFst(myfreq)

# examine the results
myfst
```

`codominant.to.dominant`

Convert a Genotype Object to Binary Data

Description

Given a genotype object in the form of a two-dimensional list of vectors, `codomiant.to.dominant` produces a matrix of binary data indicating the presence and absence of alleles in samples.

Usage

```
codominant.to.dominant(gendata, makecolinfo = FALSE, allelepresent = 1,
alleleabsent = 0, missingin = -9, missingout = -9,
loci = dimnames(gendata)[[2]], samples = dimnames(gendata)[[1]])
```

Arguments

`gendata`	Genotypes. A two dimensional list of vectors as produced by other functions in polysat. Samples are represented in the first dimension of the list and loci in the second dimension. Each vector contains all unique alleles for a given sample at a given locus.
`makecolinfo`	Boolean. If TRUE, `codominant.to.dominant` also produces a data frame indicating which loci and alleles are represented in which columns.
`allelepresent`	The symbol to be used in the output to indicate that a sample has an allele.
`alleleabsent`	The symbol to be used in the output to indicate that a sample does not have an allele.
`missingin`	The symbol used to indicate missing data in `gendata`.
`missingout`	The symbol used to indicate missing data in the output.
`loci`	A character vector that is a subset of `dimnames(gendata)[[2]]`, indicating which loci to process.
`samples`	A character vector that is a subset of `dimnames(gendata)[[1]]`, indicating which samples to process.

Value

If `makecolinfo=FALSE`, a matrix is returned containing the binary genotype data. The row names of the matrix are the sample names. The column names of the matrix are locus and allele names concatenated together, separated by a period. The symbols `allelepresent`, `alleleabsent`, and `missingout` (by default 1, 0, and -9) are used in the matrix to indicate the presence or absence of alleles, or missing data, respectively. Loci are in the same order as in the input, and alleles are arranged in numerical order.

If `makecolinfo=TRUE`, the matrix described above is returned as the first element of a list. The second element is a data frame with loci in the first column and alleles in the second column, with data in the same order as the columns in the matrix.

Author(s)

Lindsay V. Clark

See Also

`dominant.to.codominant`, `write.table`

Examples

```
# Create a data set to convert
mygendata <-
  array(list(c(124,128,138),c(122,130,140,142),c(122,132,136),c(122,134,140),
             c(203,212,218),c(197,206,221),c(215),c(200,218),
             c(140,144,148,150),c(-9),c(146,150),c(152,154,158))
        , dim=c(4,3), dimnames=list(c("ind1","ind2","ind3","ind4"),
  c("locus1","locus2","locus3")))

# Convert the data
mybinary <- codominant.to.dominant(mygendata, makecolinfo=TRUE)

# Inspect the results
mybinary[[1]]
mybinary[[2]]
```

---

distance.matrix.1locus

Pairwise Genetic Distances at One Locus

Description

Given all genotypes for one locus, create a pairwise genetic distance matrix.

Usage

```
distance.matrix.1locus(gendata, distmetric = Bruvo.distance,
progress = TRUE, ...)
```

Arguments

gendata	A sublist from a genotype object in the standard polysat format, representing all samples for one locus. A list of vectors, where each vector contains all unique alleles for a given sample at this locus. Sample names are used to name the list.
distmetric	This is the function that will be used to calculate each pairwise distance. This should be a function that, given two vectors of alleles, returns a numerical distance.
progress	If TRUE, distance.matrix.1locus will print the names of sample pairs as it finishes each calculation with distmetric. For large datasets, this is intended so that the user can monitor the progress of the calculations.
...	These arguments will be passed to distmetric. For example, with Bruvo.distance, maxl, usatnt, or missing may be used.

Details

Given a list of genotypes at one locus, distance.matrix.1locus produces a symmetrical matrix of pairwise distances between genotypes. If using a polysat genotype object such as that produced by read.GeneMapper, the gendata argument should be one of the sublists, for example mygenotypedata[,"locus1"]. The measure of distance can be any that is provided with polysat, or any function written by the user, so long as it takes genotypes as vectors of alleles (or any other type of object that is given as elements of the list gendata) and returns a numerical distance. Any arguments that need to be passed to the distmetric function can be given to distance.matrix.1locus. To save processing time, each pairwise distance is only calculated once and then written to both locations in the matrix simultaneously. The user also has the option to have each pair of sample names printed after the distance is calculated, so that progress can be monitored if evaluation is expected to take a long time.

Value

A symmetrical matrix of distances, with the names of samples used as row and column names.

Author(s)

Lindsay V. Clark

See Also

Bruvo.distance, Lynch.distance, meandistance.matrix

Examples

```
mygenotypes <- list(IND1=c(124,127,133), IND2=c(130,139,145,151),
    IND3=c(118,127,133,154))
distance.matrix.1locus(mygenotypes,usatnt=3)
```

`dominant.to.codominant` *Convert Genotypes from Dominant Format to Codominant Format*

Description

This function takes an array or matrix of genotypes where each allele is represented in one column and symbols indicate the presence or absence of that allele in a sample. It produces a two-dimensional list of vectors representing genotypes, indexed by sample and locus.

Usage

```
dominant.to.codominant(domdata, colinfo = NULL,
samples = dimnames(domdata)[[1]], missing = -9, allelepresent = 1, split
= ".")
```

Arguments

`domdata`	A two-dimensional array or matrix, in which samples are represented in the first dimension (and named accordingly) and alleles are represented in the second dimension. The symbol specified by `allelepresent` indicates that a sample has a particular allele, and any other symbol indicates that it does not. If `colinfo` is not provided by the user, the second dimension names of the array should be the locus name and allele number connected by a period or by another character as specified in `split`.
`colinfo`	A data frame, indexed by column number from `domdata`, containing locus names as the first column and allele numbers as the second column.
`samples`	A character vector containing the names of samples to be converted, if only a subset of samples in `domdata` are to be used.
`missing`	The symbol to use to represent missing data in the output.
`allelepresent`	The symbol used in `domdata` to indicate that a particular sample has a particular allele.
`split`	If `colinfo=NULL`, the character used to separate the locus name and allele number in the column names of `domdata`.

Details

Because allele copy number is often unknown, many researchers who work with microsatellites in polyploids record genotype data in a dominant format, such as a matrix of 1's and 0's to represent the presence and absence of peaks as is done with AFLPs. `dominant.to.codominant` is written to convert that data back to a semi-codominant format so that other analyses or data conversion can be performed.

The default symbol to indicate the presence of an allele is 1, but this can be set to any other symbol using the `allelepresent` argument. It does not matter which symbols are used to indicate that an allele is absent or that there is missing data. If `dominant.to.codominant` does not find any alleles present for a given sample and locus, it fills in a missing data symbol in that position in the two-dimensional genotype list.

This function does not read or write files. Since the user would already have dominant data in an array-like format in a spreadsheet or text document, it should be easily read by `read.table` and converted to a matrix by `as.matrix`.

There are two options for indicating which locus and allele is represented by each column:

1) These can be specified in the second dimension names of the array or matrix. The name of each column should be a concatenation of the locus name followed by the allele number, and these should be separated by a period or other character as specified in `split` (e.g. "locus1.204"). Note that with `check.names=TRUE`, `read.table` will convert a lot of symbols (like hyphens or spaces) to periods. It is probably a good idea to inspect the column names of `domdata` before setting `split`.

2) Create a data frame containing locus and allele information. The rows should be in the same order as the columns of `domdata`. The first vector in the data frame should contain the locus names, and the second vector in the data frame should contain the numerical alleles. Use this data frame as `colinfo`.

Value

A two-dimensional list of integer vectors, in the standard polysat genotype format. Samples are represented in the first dimension and loci in the second dimension, and both are named accordingly. Each vector contains all unique alleles for a given sample at a given locus.

Author(s)

Lindsay V. Clark

See Also

`codominant.to.dominant`, `read.table`, `as.matrix`

Examples

```
# Create a matrix of dominant data (usually read from a file instead)
mysamples <- c("ind1","ind2","ind3")
myalleles <- c("loc1.100","loc1.102","loc1.104","loc1.106",
"loc2.141","loc2.144","loc2.147","loc2.150")
mydomdata <- matrix(nrow = length(mysamples), ncol = length(myalleles),
                    dimnames = list(mysamples, myalleles))
mydomdata["ind1",] <- c(1,1,1,0,0,1,1,0)
mydomdata["ind2",] <- c(1,0,0,1,0,0,1,1)
mydomdata["ind3",] <- c(-9,-9,-9,-9,1,1,0,1)

# inspect the matrix
mydomdata

# convert to codominant data
mycodomdata <- dominant.to.codominant(mydomdata)

# view the list created
mycodomdata
# view genotypes by individual
mycodomdata["ind1",]
mycodomdata["ind2",]
mycodomdata["ind3",]
```

```
# Alternately, use a matrix without alleles labeled in the colunn names
dimnames(mydomdata)[[2]] <- NULL
mydomdata

# Make a data frame for a locus and allele index
# (Under normal circumstances you would read this from a file)
laindex <- data.frame(Loci = c(rep("loc1",4), rep("loc2",4)),
Alleles = c(100, 102, 104, 106, 141, 144, 147, 150))
laindex

# convert to codominant data
mycodomdata2 <- dominant.to.codominant(mydomdata, colinfo=laindex)
# look at the results
mycodomdata2["ind1",]
# etc.
```

`estimate.freq` *Estimate Allele Frequencies in Populations*

Description

Given genotypes, population identity, and ploidy of each individual, `estimate.freq` produces a data frame showing the estimated frequency of each allele in each population, as well as the number of genomes in each population.

Usage

```
estimate.freq(gendata, missing = -9, samples = dimnames(gendata)[[1]],
loci = dimnames(gendata)[[2]], popinfo = rep(1, length(samples)),
indploidies = rep(4, length(samples)))
```

Arguments

`gendata`	A genotype object in the standard polysat format. A two-dimensional list of vectors, where samples are represented and named in the first dimension and loci in the second dimension. Each vector contains all unique alleles for a given sample and locus.
`missing`	The symbol used to represent missing data in `gendata`.
`samples`	Character vector. The samples to be used in analysis. This should be a subset of `dimnames(gendata)[[1]]`.
`loci`	Character vector. The loci to be used in analysis. This should be a subset of `dimnames(gendata)[[2]]`.
`popinfo`	Integer or character vector. The population identity (population number or name) of each sample. The names of the vector should correspond to `samples`. If the vector is unnamed, it is assumed to be in the same order as `samples`.
`indploidies`	Integer vector. The ploidy of each sample. Should be named similarly to `popinfo`, or if unnamed is assumed to be in the same order as `samples`.

Details

This function estimates allele frequencies rather than calculating them exactly from the sample, because if there are any partially heterozygous genotypes present then allele copy number cannot be known exactly. For each sample*locus, a conversion factor is generated that is the ploidy of the sample as specified in `indploidies` divided by the number of alleles that the sample has at that locus. Each allele is then considered to be present in as many copies as the the conversion factor (note that this is not necessarily an integer). The number of copies of an allele is totaled for the population and is divided by the total number of genomes in the population (minus missing data at the locus) in order to calculate allele frequency.

A major assumption of this calculation method is that each allele in a partially heterozygous genotype has an equal chance of being present in more than one copy. This is almost never true, because common alleles in a population are more likely to be partially homozygous in an individual. The result is that the frequency of common alleles is underestimated and the frequency of rare alleles is overestimated.

Value

Data frame, where each population is in one row. The first column is called `Genomes` and contains the number of genomes in each population. Each remaining column contains frequencies for one allele. Columns are named by locus and allele, separated by a period.

Author(s)

Lindsay V. Clark

See Also

`calcFst`

Examples

```
# create a data set (typically done by reading files)
mygenotypes <- array(list(-9), dim = c(6,2), dimnames =
                     list(paste("ind",1:6, sep=""),c("loc1","loc2")))
mygenotypes[,"loc1"] <- list(c(206),c(208,210),c(204,206,210),
    c(196,198,202,208),c(196,200),c(198,200,202,204))
mygenotypes[,"loc2"] <- list(c(130,134),c(138,140),c(130,136,140),
    c(138),c(136,140),c(130,132,136))

mypopinfo <- c(1,1,1,2,2,2)
names(mypopinfo) <- dimnames(mygenotypes)[[1]]

myploidies <- c(2,2,4,4,2,4)
names(myploidies) <- dimnames(mygenotypes)[[1]]

# calculate allele frequencies
myfreq <- estimate.freq(mygenotypes, popinfo=mypopinfo,
indploidies=myploidies)

# look at the results
myfreq
```

`estimate.ploidy` *Maximum and Mean Allele Count for Estimation of Ploidy*

Description

Given genotypes in the form of a two-dimensional list of vectors, estimate.ploidy produces a two-dimensional array containing the maximum number of alleles and the mean number of alleles for each sample, across all loci.

Usage

```
estimate.ploidy(gendata, samples = dimnames(gendata)[[1]],
loci = dimnames(gendata)[[2]])
```

Arguments

`gendata`	Genotypes in the standard polysat format. A two-dimensional list of vectors, where samples are represented and named in the first dimension and loci in the second dimension. Each vector contains all unique alleles for a given sample and locus.
`samples`	An optional character vector of samples to evaluate, which is a subset of `dimnames(gendata)[[1`
`loci`	An optional character vector of loci to use in the calculation, which is a subset of `dimnames(gendata)[[2]]`.

Details

To assist the user in determining the ploidy of each sample, `estimate.ploidy` looks at the genotype of the sample across all loci and returns the maximum number of alleles per locus. The mean number of alleles is also returned to assist with checking for errors (for example, if an octoploid genotype was accidentally scored at one locus for a diploid sample). Both of these are calculated using the `length` function on the genotype vectors.

The user may want to extract the vector containing the maximum number of alleles (for example, `myploidies<-ploidyinfo[,1]`) and then manually edit the values based on other knowledge of the organism. This vector can then be used as the `indploidies` argument for `write.Structure` or `estimate.freq`.

Value

An array with the second dimension of length 2 and the first dimension as long as `samples`. The rows are labeled by sample name and the columns are labeled `max.alleles` and `mean.alleles`.

Author(s)

Lindsay V. Clark

See Also

`estimate.freq`, `write.Structure`

Examples

```
# Create a data set to analyze
mygendata <-
  array(list(c(124,128,138),c(122,130,140,142),c(122,132,136),
             c(122,134,140),
             c(203,212,218),c(197,206,221),c(215),c(200,218),
             c(140,144,148,150),c(-9),c(146,150),c(152,154,158))
        , dim=c(4,3), dimnames=list(c("ind1","ind2","ind3","ind4"),
        c("locus1","locus2","locus3")))

# Run the function
estimate.ploidy(mygendata)
```

FCRinfo *Additional Data on Rubus Samples*

Description

For 20 Rubus samples, contains numbers indicating species identity, as well as colors and symbols to use for plotting data.

Usage

```
data(FCRinfo)
```

Format

Data frame. `FCRinfo$Species` contains integers indicating the species of each sample. `FCRinfo$Plot.color` contains character strings of the colors that each sample should be plotted as. `FCR.info$Plot.symbol` contains integers to be passed to `pch` to designate the symbol used to represent each individual.

Source

Clark and Jasieniuk, unpublished data

See Also

`testgenotypes`

---

find.missing.gen *Find Missing Genotypes*

Description

This function returns a data frame listing the locus and sample names of all genotypes with missing data.

Usage

```
find.missing.gen(gendata, missing = -9, samples =
  dimnames(gendata)[[1]], loci = dimnames(gendata)[[2]])
```

Arguments

gendata	A genotype object in the standard polysat format. A two dimensional list of vectors, where samples are represented and named in the first dimension and loci in the second dimension. Each vector contains all unique alleles for a given sample and locus. If data is missing, the missing data symbol is the first element of the vector.
missing	The symbol used to indicate missing data.
samples	A character vector of all samples to be searched. Must be a subset of `dimnames(gendata)[[1]]`
loci	A character vector of all loci to be searched. Must be a subset of `dimnames(gendata)[[2]]`.

Value

A data frame with no row names. The first column is named "Locus" and the second column is named "Sample". Each row represents one missing genotype, and gives the locus and sample of that genotype.

Author(s)

Lindsay V. Clark

Examples

```
# set up the genotype data
samples <- paste("ind", 1:4, sep="")
samples
loci <- paste("loc", 1:3, sep="")
loci
testgen <- array(list(-9), dim = c(4,3), dimnames = list(samples,loci))
testgen[,"loc1"] <- list(c(-9), c(102,104), c(100,106,108,110,114),
                         c(102,104,106,110,112))
testgen[,"loc2"] <- list(c(77,79,83), c(79,85), c(-9), c(83,85,87,91))
testgen[,"loc3"] <- list(c(122,128), c(124,126,128,132), c(120,126),
                         c(124,128,130))

# look up which samples*loci have missing genotypes
find.missing.gen(testgen)
```

Lynch.distance *Calculate Band-Sharing Dissimilarity Between Genotypes*

Description

Given two genotypes in the form of vectors of unique alleles, a dissimilarity is calculated as: 1 - (number of alleles in common)/(average number of alleles per genotype).

Usage

```
Lynch.distance(genotype1, genotype2, missing = -9)
```

Arguments

genotype1	A vector containing all alleles for a particular sample and locus. Each allele is only present once in the vector.
genotype2	A vector of the same form as genotype1, for another sample at the same locus.
missing	The symbol used to indicate missing data in either genotype vector.

Details

Lynch (1990) defines a simple measure of similarity between DNA fingerprints. This is 2 times the number of bands that two fingerprints have in common, divided by the total number of bands that the two genotypes have. Lynch.distance returns a dissimilarity, which is 1 minus the similarity.

Value

If the first element of either or both input genotypes is equal to missing, NA is returned.

Otherwise, a numerical value is returned. This is one minus the similarity. The similarity is calculated as the number of alleles that the two genotypes have in common divided by the mean length of the two genotypes.

Author(s)

Lindsay V. Clark

References

Lynch, M. (1990) The Similarity Index and DNA Fingerprinting. *Molecular Biology and Evolution* **7**, 478-484.

See Also

Bruvo.distance, distance.matrix.1locus, meandistance.matrix

Examples

```
Lynch.distance(c(100,102,104), c(100,104,108))
Lynch.distance(-9, c(102,104,110))
Lynch.distance(c(100), c(100,104,106))
```

---

meandist.from.array

Tools for Working With Pairwise Distance Arrays

Description

meandist.from.array produces a mean distance matrix from an array of pairwise distances by locus, such as that produced by meandistance.matrix when all.distances=TRUE. find.na.dist finds missing distances in such an array, and find.na.dist.not.missing finds missing distances that aren't the result of missing genotypes.

Usage

```
meandist.from.array(distarray, samples = dimnames(distarray)[[2]],
loci = dimnames(distarray)[[1]])

find.na.dist(distarray, samples = dimnames(distarray)[[2]],
loci = dimnames(distarray)[[1]])

find.na.dist.not.missing(gendata, distarray, missing = -9,
samples = dimnames(distarray)[[2]], loci = dimnames(distarray)[[1]])
```

Arguments

distarray	A three-dimensional array of pairwise distances between samples, by locus. Loci are represented in the first dimension, and samples are represented in the second and third dimensions. Dimensions are named accordingly. Such an array is the first element of the list produced by meandistance.matrix if all.distances=TRUE.
samples	Character vector. Samples to analyze.
loci	Character vector. Loci to analyze.
gendata	A genotype object in the standard polysat format. Typically the genotype object that was used to produce distarray.
missing	The symbol used to represent missing data in gendata.

Details

find.na.dist.not.missing is primarily intended to locate distances that were not calculated by Bruvo.distance because both genotypes had too many alleles (more than maxl). The user may wish to estimate these distances manually and fill them into the array, then recalculate the mean matrix using meandist.from.array.

Value

meandist.from.array returns a matrix, with both rows and columns named by samples, of distances averaged across loci.

find.na.dist and find.na.dist.not.missing both return data frames with three columns: Locus, Sample1, and Sample2. Each row represents the index in the array of an element containing NA.

Author(s)

Lindsay V. Clark

See Also

meandistance.matrix, Bruvo.distance

Examples

```
# set up the genotype data
samples <- paste("ind", 1:4, sep="")
samples
loci <- paste("loc", 1:3, sep="")
```

```
loci
testgen <- array(list(-9), dim = c(4,3), dimnames = list(samples,loci))
testgen[,"loc1"] <- list(c(-9), c(102,104), c(100,106,108,110,114),
                         c(102,104,106,110,112))
testgen[,"loc2"] <- list(c(77,79,83), c(79,85), c(-9), c(83,85,87,91))
testgen[,"loc3"] <- list(c(122,128), c(124,126,128,132), c(120,126),
                         c(124,128,130))

# look up which samples*loci have missing genotypes
find.missing.gen(testgen)

# get the three-dimensional distance array and the mean of the array
gendist <- meandistance.matrix(testgen, distmetric=Bruvo.distance,
                               maxl=4, all.distances=TRUE)
# look at the distances for loc1, where there is missing data and long genotypes
gendist[[1]]["loc1",,]

# look up all missing distances in the array
find.na.dist(gendist[[1]])

# look up just the missing distances that don't result from missing genotypes
find.na.dist.not.missing(testgen, gendist[[1]])

# Copy the array to edit the new copy
newDistArray <- gendist[[1]]
# calculate the distances that were NA from genotype lengths exceeding maxl
# (in reality, if this were too computationally intensive you might estimate
# it manually instead)
subDist <- Bruvo.distance(c(100,106,108,110,114), c(102,104,106,110,112))
subDist
# insert this distance into the correct positions
newDistArray["loc1","ind3","ind4"] <- subDist
newDistArray["loc1","ind4","ind3"] <- subDist
# calculate the new mean distance matrix
newMeanMatrix <- meandist.from.array(newDistArray)
# look at the difference between this matrix and the original.
newMeanMatrix
gendist[[2]]
```

`meandistance.matrix`

Mean Pairwise Distance Matrix

Description

Given a two-dimensional list of genotypes, indexed by sample and locus, `meandistance.matrix` produces a symmetrical matrix of pairwise distances between samples, averaged across all loci. An array of all distances prior to averaging may also be produced.

Usage

```
meandistance.matrix(gendata, samples=dimnames(gendata)[[1]],
loci=dimnames(gendata)[[2]], all.distances=FALSE, usatnts=NULL, ...)
```

Arguments

gendata	A two-dimensional list of genotypes in the standard polysat format. The first dimension is an index of samples, the second dimension is an index of loci, and the elements are numerical vectors containing the alleles as elements.
samples	A character vector of samples to be analyzed. These should be all or a subset of the sample names used in gendata.
loci	A character vector of loci to be analyzed. These should be all or a subset of the loci names used in gendata.
all.distances	If FALSE, only the mean distance matrix will be returned. If TRUE, a list will be returned containing an array of all distances by locus and sample as well as the mean distance matrix.
usatnts	A numerical vector that contains the length of nucleotide repeats for each locus. For example, 3 would be used to indicate a locus with trinucleotide repeats. 1 should be used if alleles are written in terms of repeat number, not fragment length in nucleotides. names(usatnts) should be the same as those used in dimnames(gendata)[[2]] (the names of the loci). This argument can be omitted if repeat length is irrelevant to the distance metric.
...	If distmetric or progress are given here they will be passed to distance.matrix.1locus Any other arguments will be passed to distmetric.

Details

meandistance.matrix uses distance.matrix.1locus once for each locus to be analyzed, then averages values across these matrices. Any arguments that need to be passed to distance.matrix.1locus may be given to meandistance.matrix. If the loci are of different repeat types and the type of repeat is important for the distance metric being used (e.g. Bruvo.distance), the usatnts argument can be used to pass a different usatnt argument to distmetric depending on the locus.

Because the user may want to omit samples or loci, the samples and loci arguments are given for convenient indexing of the data to be analyzed. If gendata contains only the data that the user wants to analyze, the user can simply omit these arguments.

Missing data must be represented by the missing data symbol, rather than NA.

Value

A symmetrical matrix containing pairwise distances between all samples, averaged across all loci. Row and column names of the matrix will be the sample names provided in the 'samples' argument. If all.distances=TRUE, a list will be produced containing the above matrix as well as a three-dimensional array containing all distances by locus and sample. The array is the first item in the list, and the mean matrix is the second.

Author(s)

Lindsay V. Clark

See Also

distance.matrix.1locus, Bruvo.distance, Lynch.distance

Examples

```
# create a list of genotype data
mygendata <-
  array(list(c(124,128,138),c(122,130,140,142),c(122,132,136),c(122,134,140),
             c(203,212,218),c(197,206,221),c(215),c(200,218),
             c(140,144,148,150),c(-9),c(146,150),c(152,154,158),
             c(233,236,280),c(-9),c(-9),c(-9))
        ,dim=c(4,4),dimnames=list(c("ind1","ind2","ind3","ind4"),
                                  c("locus1","locus2","locus3","locus4")))

# make index vectors of data to use
myloci <- c("locus1","locus2","locus3")
mysamples <- c("ind1","ind2","ind4")

# locus1 and locus3 have dinucleotide repeats, and locus2 has
# trinucleotide repeats
myusatnts <- c(2,3,2)
names(myusatnts) <- myloci

meandistance.matrix(mygendata, mysamples, myloci, all.distances=TRUE,
                    usatnts=myusatnts)
```

---

read.ATetra *Read File in ATetra Format*

Description

Given a file formatted for the software ATetra, `read.ATetra` produces a vector of the population identities of each sample, as well as a two-dimensional list of vectors representing the genotypes.

Usage

```
read.ATetra(infile)
```

Arguments

`infile`	Character string. A file path to the file to be read.

Details

`read.ATetra` reads text files in the exact format specified by the ATetra documentation. Note that this format only allows tetraploid (or lower) data and that there can be no missing data. The genotype object produced is in the same format that is produced or read by other functions in the polysat package.

Value

`PopData`	A numerical vector containing the population number for each sample (individual). The sample names are used as the names of this vector.
`Genotypes`	A two dimensional list of integer vectors in the standard polysat genotype format. The first dimension represents samples and the second represents loci, and both are named accordingly. Each vector contains as elements the alleles for that sample at that locus.

Author(s)

Lindsay V. Clark

References

http://www.vub.ac.be/APNA/ATetra_Manual-1-1.pdf

van Puyvelde, K., van Geert, A. and Triest, L. (2010) ATETRA, a new software program to analyze tetraploid microsatellite data: comparison with TETRA and TETRASAT. *Molecular Ecology Resources* **10**, 331-334.

See Also

read.Tetrasat, read.GeneMapper, read.Structure, read.GenoDive, dominant.to.codominant
write.ATetra, read.SPAGeDi

Examples

```
# create a file to be read
# (this would normally be done in a text editor or with ATetra's Excel template)
cat("TIT,Sample Rubus Data for ATetra", "LOC,1,CBA15",
"POP,1,1,Commonwealth", "IND,1,1,1,CMW1,197,208,211,213",
"IND,1,1,2,CMW2,197,207,211,212", "IND,1,1,3,CMW3,197,208,212,219",
"IND,1,1,4,CMW4,197,208,212,219", "IND,1,1,5,CMW5,197,208,211,212",
"POP,1,2,Fall Creek Lake", "IND,1,2,6,FCR4,197,207,211,212",
"IND,1,2,7,FCR7,197,208,212,218", "IND,1,2,8,FCR14,197,207,212,218",
"IND,1,2,9,FCR15,197,208,211,212", "IND,1,2,10,FCR16,197,208,211,212",
"IND,1,2,11,FCR17,197,207,212,218","LOC,2,CBA23","POP,2,1,Commonwealth",
"IND,2,1,1,CMW1,98,100,106,125","IND,2,1,2,CMW2,98,125,,",
"IND,2,1,3,CMW3,98,126,,","IND,2,1,4,CMW4,98,106,119,127",
"IND,2,1,5,CMW5,98,106,125,","POP,2,2,Fall Creek Lake",
"IND,2,2,6,FCR4,98,125,,","IND,2,2,7,FCR7,98,106,126,",
"IND,2,2,8,FCR14,98,127,,","IND,2,2,9,FCR15,98,108,117,",
"IND,2,2,10,FCR16,98,125,,","IND,2,2,11,FCR17,98,126,,","END",
file = "atetraexample.txt", sep = "\n")

# Read the file and examine the data
exampledata <- read.ATetra("atetraexample.txt")
exampledata$PopData
exampledata$Genotypes
exampledata$Genotypes[["CMW1","CBA23"]]
```

---

read.GeneMapper *Read GeneMapper Genotypes Tables*

Description

Given a vector of filepaths to tab-delimited text files containing genotype data in the ABI GeneMapper Genotypes Table format, read.GeneMapper produces a two-dimensional list of genotypes that can be read by other functions in the polysat package.

Usage

```
read.GeneMapper(infiles, missing = -9)
```

Arguments

infiles	A character vector of paths to the files to be read.
missing	A numerical value used to indicate missing data for a given sample and locus.

Details

read.GeneMapper can read the genotypes tables that are exported by the Applied Biosystems GeneMapper software. The only alterations to the files that the user may have to make are 1) delete any rows with missing data or fill in a numerical missing data symbol of your choice (such as -9) in the first allele slot for that row, 2) make sure that all allele names are numeric representations of fragment length (no question marks or dashes), and 3) put sample names into the Sample Name column, if the names that you wish to use in analysis are not already there. Each file should have the standard header row produced by the software. If any sample has more than one genotype listed for a given locus, only the last genotype listed will be used.

The file format is simple enough that the user can easily create files manually if GeneMapper is not the software used in allele calling. The files are tab-delimited text files. There should be a header row with column names. The column labeled "Sample Name" should contain the names of the samples, and the column labeled "Marker" should contain the names of the loci. You can have as many or as few columns as needed to contain the alleles, and each of these columns should be labeled "Allele X" where X is a number unique to each column. Row labels and any other columns are ignored. For any given sample, each allele is listed only once and is given as an integer that is the length of the fragment in nucleotides. Alleles are separated by tabs. If you have more allele columns than alleles for any given sample, leave the extra cells blank so that read.table will read them as NA. Example data files in this format are included in the package.

read.GeneMapper will read all of your data at once. It takes as its first argument a character vector containing paths to all of the files to be read. How the data are distributed over these files does not matter. The function finds all unique sample names and all unique markers across all the files, and automatically puts a missing data symbol into the list if a particular sample and locus combination is not found. Rows in which all allele cells are blank should NOT be included in the input files; either delete these rows or put the missing data symbol into the first allele cell.

Sample and locus names must be consistent within and across the files. The list that is produced is indexed by these names. For example, if the object produced was called mygenotypes, mygenotypes[["AB1","ABC5"]] would be a vector containing alleles for sample AB1 at locus ABC5. mygenotypes[,"ABC5"] would be a list of all genotypes at locus ABC5, and mygenotypes["AB1",] would be a list of all genotypes for sample AB1.

Value

The object produced is a two-dimensional list of vectors representing the genotypes. Samples are represented by the first dimension and loci by the second dimension. The names of the samples and loci are taken from the Sample Names and Markers columns, respectively, of the GeneMapper files. The vectors at each position of the list are numerical and are only as long as needed to contain each allele (for that sample and locus) as one element.

Note

A 'subscript out of bounds' error may mean that a sample name or marker was left blank in one of the input files.

Author(s)

Lindsay V. Clark

References

http://www.appliedbiosystems.com/genemapper

See Also

read.Tetrasat, read.ATetra, read.Structure, read.GenoDive, write.GeneMapper, dominant.to.codominant, read.SPAGeDi

Examples

```
  ## Not run:
    myinfiles<-c("data\sample CBA15.txt","data\sample
    CBA23.txt","data\sample CBA28.txt")
    mygenotypes<-read.GeneMapper(myinfiles)

    #Look at the object produced.  Alleles are not listed but you can
    #see that the array was filled.
    mygenotypes

    #Look at the genotype of individual FCR5.
    mygenotypes["FCR5",]

    #Correct one of the genotypes
    mygenotypes[["FCR5","RhCBA15"]]<-c(208)

## End(Not run)

# an example with defined data:
# create a table of data
gentable <- data.frame(Sample.Name=rep(c("ind1","ind2","ind3"),2),
                       Marker=rep(c("loc1","loc2"), each=3),
                       Allele.1=c(202,200,204,133,133,130),
                       Allele.2=c(206,202,208,136,142,136),
                       Allele.3=c(NA,208,212,145,148,NA),
                       Allele.4=c(NA,216,NA,151,157,NA)
                       )
# create a file (inspect this file in a text editor or spreadsheet
# software to see the required format)
write.table(gentable, file="readGMtest.txt", quote=FALSE, sep="\t",
            na="", row.names=FALSE, col.names=TRUE)

# read the file
mygenotypes <- read.GeneMapper("readGMtest.txt")

# inspect the results
mygenotypes[,"loc1"]
mygenotypes[,"loc2"]
```

`read.GenoDive` *Import Genotype Data from GenoDive File*

Description

`read.GenoDive` takes a text file in the format for the software GenoDive and produces a vector indicating which samples are in which populations and a two-dimensional list of vectors containing the genotypes.

Usage

```
read.GenoDive(infile, missing = -9)
```

Arguments

`infile`	A character string. The path to the file to be read.
`missing`	The symbol used to represent missing data in the output.

Details

GenoDive is a Mac-only program for population genetic analysis that allows for polyploid data. `read.GenoDive` imports data from text files formatted for this program.

The first line of the file is a comment line, ignored by the function. On the second line, separated by tabs, are the number of individuals, number of populations, number of loci, maximum ploidy (ignored), and number of digits used to code alleles.

The following lines contain the names of populations, which are ignored by the function although the number of lines should be the same as the number of populations as specified in the first line. After that is a header line for the genotype data. This line contains, separated by tabs, column headers for populations, clones (optional), and individuals, followed by the name of each locus. The loci names for the genotype object are derived from this line.

Each individual is on one line following the genotype header line. Separated by tabs are the population number, the clone number (optional), the individual name (used as the sample name in the output) and the genotypes at each locus. Alleles at one locus are concatenated together in one string without any characters to separate them. Each allele must have the same number of digits, although leading zeros can be omitted.

If the only alleles listed for a particular individual and locus are zeros, this is interpreted by `read.GenoDive` as missing data, and `missing` (-9 by default) is recorded in the list. GenoDive allows for a genotype to be partially missing but polysat does not; therefore, if an allele is coded as zero but other alleles are recorded for that sample and locus, the output genotype will just contain the alleles that are present, with the zeros thrown out.

Value

`PopData`	A numeric vector containing the population number of each individual. The names of the vector are the names of the individuals.
`Genotypes`	A two-dimensional list of integer vectors, containing genotypes in the same format that is produced and read by other polysat functions. The first dimension is indexed by sample and the second dimension is indexed by locus. Each vector contains all unique alleles for a given sample and locus.

Author(s)

Lindsay V. Clark

References

Meirmans, P. G. and Van Tienderen, P. H. (2004) GENOTYPE and GENODIVE: two programs for the analysis of genetic diversity of asexual organisms. *Molecular Ecology Notes* **4**, 792-794.

http://www.bentleydrummer.nl/software/software/GenoDive.html

See Also

read.GeneMapper, write.GenoDive, read.Tetrasat, read.ATetra, read.Structure, dominant.to.codominant, read.SPAGeDi

Examples

```
# create data file (normally done in a text editor or spreadsheet software)
cat(c("example comment line", "5\t2\t2\t3\t2", "pop1", "pop2",
      "pop\tind\tloc1\tloc2", "1\tJohn\t102\t1214",
      "1\tPaul\t202\t0", "2\tGeorge\t101\t121213",
      "2\tRingo\t10304\t131414","1\tYoko\t10303\t120014"),
    file = "genodiveExample.txt", sep = "\n")

# import file data
exampledata <- read.GenoDive("genodiveExample.txt")

# view data
exampledata$PopData
exampledata$Genotypes[,"loc1"]
exampledata$Genotypes[,"loc2"]
```

---

read.SPAGeDi *Read Genotypes in SPAGeDi format*

Description

read.SPAGeDi can read a text file formatted for the SPAGeDi software and return a genotype object in the standard polysat format, as well as optionally returning a vector of individual ploidies and/or a data frame of categories and spatial coordinates.

Usage

```
read.SPAGeDi(infile, allelesep = "/", returncatspatcoord = FALSE,
             returnploidies = FALSE, missing = -9)
```

Arguments

infile	Character string. Path of the file to read.
allelesep	The character that is used to delimit alleles within genotypes, or "" if alleles have a fixed number of digits and are not delimited by any character. Other examples shown in section 3.2.1 of the SPAGeDi 1.3 manual include "/", " ", ", ", ".", and "--".
returncatspatcoord	Boolean. Indicates whether a data frame should be returned containing the category and spatial coordinates columns.

`returnploidies`
: Boolean. Indicates whether a vector should be returned containing the ploidy of each individual.

`missing`
: The symbol to be used to specify missing data in the genotype object that is returned.

Details

SPAGeDi offers a lot of flexibility in how data files are formatted. `read.SPAGeDi` accomodates most of that flexibility. The primary exception is that alleles must be delimited in the same way across all genotypes, as specified by `allelesep`. Comment lines beginning with `//`, as well as blank lines, are ignored by `read.SPAGeDi` just as they are by SPAGeDi.

`read.SPAGeDi` is not designed to read dominant data (see section 3.2.2 of the SPAGeDi 1.3 manual). However, see `dominant.to.codominant` for a way to read this type of data after some simple manipulation in a spreadsheet program.

The first line of a SPAGeDi file contains information that is used by `read.SPAGeDi`. The ploidy as specified in the 6th position of the first line is ignored, and is instead calculated by counting alleles for each individual (including zeros on the right, but not the left, side of the genotype). The number of digits specified in the 5th position of the first line is only used if `allelesep=""`. All other values in the first line are important for the function.

If the only alleles found for a particular individual and locus are zeros, the genotype is interpreted as missing. Otherwise, zeros on the left side of a genotype are ignored, and zeros on the right side of a genotype are used in calculating the ploidy but are not included in the genotype object that is returned. If `allelesep=""`, `read.SPAGeDi` checks that the number of characters in the genotype can be evenly divided by the number of digits per allele. If not, zeros are added to the left of the genotype string before splitting it into alleles.

Value

Under the default where `returncatspatcoord=FALSE` and `returnploidies=FALSE`, a genotype object in the standard polysat format is returned. This is a two-dimensional list of integer vectors, where the first dimension of the list represents samples and the second dimension of the list represents loci. Both dimensions are named by the individual and locus names found in the file. Each vector contains all unique alleles, formatted as integers.

Otherwise, a list of two or three objects is returned:

`CatSpatCoord`
: A data frame of categories and spatial coordinates, unchanged from the file. The format of each column is determined under the default `read.table` settings. Row names are individual names from the file. Column names are the same as in the file.

`Indploidies`
: A vector containing the ploidy of each individual, as determined by the (maximum) number of alleles per genotype, including zeros on the right. The vector is named by the individual names from the file.

`Genotypes`
: A genotype object as described above.

Author(s)

Lindsay V. Clark

References

http://ebe.ulb.ac.be/ebe/Software_files/manual_SPAGeDi_1-3.pdf

Hardy, O. J. and Vekemans, X. (2002) SPAGeDi: a versatile computer program to analyse spatial genetic structure at the individual or population levels. *Molecular Ecology Notes* **2**, 618-620.

See Also

write.SPAGeDi, dominant.to.codominant, read.table, read.GeneMapper, read.GenoDive, read.Structure, read.ATetra, read.Tetrasat

Examples

```
# create a file to read (usually done with spreadsheet software or a
# text editor):
cat("// here's a comment line at the beginning of the file",
"5\t0\t-2\t2\t2\t4",
"4\t5\t10\t50\t100",
"Ind\tLat\tLong\tloc1\tloc2",
"ind1\t39.5\t-120.8\t00003133\t00004040",
"ind2\t39.5\t-120.8\t3537\t4246",
"ind3\t42.6\t-121.1\t5083332\t40414500",
"ind4\t38.2\t-120.3\t00000000\t41430000",
"ind5\t38.2\t-120.3\t00053137\t00414200",
"END",
sep="\n", file="SpagInputExample.txt")

# display the file
cat(readLines("SpagInputExample.txt"), sep="\n")

# read the file
mydata <- read.SPAGeDi("SpagInputExample.txt", allelesep = "",
returncatspatcoord = TRUE, returnploidies = TRUE)

# view the data
mydata

# view genotypes by locus
mydata$Genotypes[,"loc1"]
mydata$Genotypes[,"loc2"]
```

---

read.Structure *Read Genotypes and Other Data from a Structure File*

Description

read.Structure creates a genotype object of the standard polysat format by reading a text file formatted for the software Structure. Optionally, it can also extract PopData and any other extra columns from the file.

Usage

```
read.Structure(infile, missingin = -9, missingout = -9, sep = "\t",
markernames = TRUE, labels = TRUE, extrarows = 1, extracols = 0,
ploidy = 4, getexcols = FALSE)
```

Arguments

infile	Character string. The file path to be read.
missingin	The symbol used to represent missing data in the Structure file.
missingout	The symbol to be used to represent missing data in the genotype object that is produced.
sep	The character used to delimit the fields of the Structure file (tab by default).
markernames	Boolean, indicating whether the file has a header containing marker names.
labels	Boolean, indicating whether the file has a column containing sample names.
extrarows	Integer. The number of extra rows that the file has, not counting marker names. This could include rows for recessive alleles, inter-marker distances, or phase information.
extracols	Integer. The number of extra columns that the file has, not counting sample names (labels). This could include PopData, PopFlag, LocData, Phenotype, or any other extra columns.
ploidy	Integer. The ploidy of the file, i.e. how many rows there are for each individual.
getexcols	Boolean, indicating whether the function should return the data from any extra columns.

Details

The current version of `read.Structure` does not support the ONEROWPERIND option in the file format. Each locus must only have one column. If your data is in ONEROWPERIND format, it should be fairly simple to manipulate it in a spreadsheet program so that it can be read by `read.GeneMapper` instead.

`read.Structure` uses `read.table` to initially read the file into a data frame, then extracts information from the data frame. Because of this, any header rows (particularly the one containing marker names) should have leading tabs (or spaces if `sep=" "`) so that the marker names align correctly with their corresponding genotypes. You should be able to open the file in a spreadsheet program and have everything align correctly.

If the file does not contain sample names, set `labels=FALSE`. The samples will be numbered instead, and if you like you can edit the `dimnames` of the genotype object after the fact if you have the sample names stored separately. Likewise, if `markernames=FALSE`, the loci will be numbered automatically by the column names that `read.table` creates, but these can also be edited after the fact.

Value

If `getexcols=FALSE`, the function returns only a genotype object in the standard polysat format. This is a two-dimensional list, where samples are represented in the first dimension and loci in the second dimension. Each element of the list is an integer vector containing all unique alleles for a given locus and sample.

If `getexcols=TRUE`, the function returns a list with two elements. The first, named ExtraCol, is a data frame, where the row names are the sample names and each column is one of the extra columns from the file (but with each sample only once instead of being repeated `ploidy` number of times). The second element is named Genotypes and is the genotype object described above.

Author(s)

Lindsay V. Clark

References

http://pritch.bsd.uchicago.edu/structure_software/release_versions/v2.3.3/structure_doc.pdf

Hubisz, M. J., Falush, D., Stephens, M. and Pritchard, J. K. (2009) Inferring weak population structure with the assistance of sample group information. *Molecular Ecology Resources* **9**, 1322-1332.

Falush, D., Stephens, M. and Pritchard, J. K. (2007) Inferences of population structure using multilocus genotype data: dominant markers and null alleles. *Molecular Ecology Notes* **7**, 574-578.

See Also

write.Structure, read.GeneMapper, read.Tetrasat, read.ATetra, read.GenoDive, dominant.to.codominant, read.SPAGeDi

Examples

```
# create a file to read (normally done in a text editor or spreadsheet
# software)
cat("\t\tRhCBA15\tRhCBA23\tRhCBA28\tRhCBA14\tRUB126\tRUB262\tRhCBA6\tRUB26",
    "\t\t-9\t-9\t-9\t-9\t-9\t-9\t-9\t-9",
    "WIN1B\t1\t197\t98\t152\t170\t136\t208\t151\t99",
    "WIN1B\t1\t208\t106\t174\t180\t166\t208\t164\t99",
    "WIN1B\t1\t211\t98\t182\t187\t184\t208\t174\t99",
    "WIN1B\t1\t212\t98\t193\t170\t203\t208\t151\t99",
    "WIN1B\t1\t-9\t-9\t-9\t-9\t-9\t-9\t-9\t-9",
    "WIN1B\t1\t-9\t-9\t-9\t-9\t-9\t-9\t-9\t-9",
    "WIN1B\t1\t-9\t-9\t-9\t-9\t-9\t-9\t-9\t-9",
    "WIN1B\t1\t-9\t-9\t-9\t-9\t-9\t-9\t-9\t-9",
    "MCD1\t2\t208\t100\t138\t160\t127\t202\t151\t124",
    "MCD1\t2\t208\t102\t153\t168\t138\t207\t151\t134",
    "MCD1\t2\t208\t106\t157\t180\t162\t211\t151\t137",
    "MCD1\t2\t208\t110\t159\t187\t127\t215\t151\t124",
    "MCD1\t2\t208\t114\t168\t160\t127\t224\t151\t124",
    "MCD1\t2\t208\t124\t193\t160\t127\t228\t151\t124",
    "MCD1\t2\t-9\t-9\t-9\t-9\t-9\t-9\t-9\t-9",
    "MCD1\t2\t-9\t-9\t-9\t-9\t-9\t-9\t-9\t-9",
    "MCD2\t2\t208\t98\t138\t160\t136\t202\t150\t120",
    "MCD2\t2\t208\t102\t144\t174\t145\t214\t150\t132",
    "MCD2\t2\t208\t105\t148\t178\t136\t217\t150\t135",
    "MCD2\t2\t208\t114\t151\t184\t136\t227\t150\t120",
    "MCD2\t2\t208\t98\t155\t160\t136\t202\t150\t120",
    "MCD2\t2\t208\t98\t157\t160\t136\t202\t150\t120",
    "MCD2\t2\t208\t98\t163\t160\t136\t202\t150\t120",
    "MCD2\t2\t208\t98\t138\t160\t136\t202\t150\t120",
    "MCD3\t2\t197\t100\t172\t170\t159\t213\t174\t134",
    "MCD3\t2\t197\t106\t174\t178\t193\t213\t176\t132",
    "MCD3\t2\t-9\t-9\t-9\t-9\t-9\t-9\t-9\t-9",
    "MCD3\t2\t-9\t-9\t-9\t-9\t-9\t-9\t-9\t-9",
    "MCD3\t2\t-9\t-9\t-9\t-9\t-9\t-9\t-9\t-9",
    "MCD3\t2\t-9\t-9\t-9\t-9\t-9\t-9\t-9\t-9",
    "MCD3\t2\t-9\t-9\t-9\t-9\t-9\t-9\t-9\t-9",
    "MCD3\t2\t-9\t-9\t-9\t-9\t-9\t-9\t-9\t-9",
    sep="\n",file="structtest.txt")

# view the file
```

```
cat(readLines("structtest.txt"), sep="\n")

# read the structure file into genotypes and populations
testdata <- read.Structure("structtest.txt", extracols=1,
                           ploidy=8, getexcols=TRUE)

# examine the results
testdata$ExtraCol
testdata$Genotypes
testdata$Genotypes["WIN1B",]
```

`read.Tetrasat` *Read Data from a TETRASAT Input File*

Description

Given a file containing genotypes in the TETRASAT format, `read.Tetrasat` produces a vector containing the population identity of each individual as well as a two-dimensional list of vectors containing the genotypes.

Usage

```
read.Tetrasat(infile, missing = -9)
```

Arguments

`infile`	A character string of the file path to be read.
`missing`	A numerical value to be used to indicate missing data in the object produced by the function.

Details

`read.Tetrasat` reads text files that are in the exact format specified by the software TETRASAT and TETRA (see references for more information). This is similar to the file format for GenePop but allows for up to four alleles per locus. All alleles must be coded by two digits. Another diference between the TETRASAT and GenePop formats is that in TETRASAT the sample name and genotypes are not separated by a comma, because the columns of data have fixed widths.

Since TETRASAT files also contain information about which samples belong to which populations, a vector is also produced containing this information. The elements of the vector are numbers representing the populations, and the names of the elements are the sample names. This can be used as the popinfo argument for `estimate.freq`, for example, or converted to a column in an array for use in the `extracols` argument of `write.Structure`.

Value

`PopData`	An integer vector containing the population ID for each sample. Sample names are used as the names of the elements.
`Genotypes`	A two-dimensional list of integer vectors, in the standard polysat genotype format. Samples are the first dimension, and loci are the second dimension. All white space is stripped from sample names. Each vector contains all unique alleles for that sample at that locus.

Author(s)

Lindsay V. Clark

References

http://markwith.freehomepage.com/tetrasat.html

Markwith, S. H., Stewart, D. J. and Dyer, J. L. (2006) TETRASAT: a program for the population analysis of allotetraploid microsatellite data. *Molecular Ecology Notes* **6**, 586-589.

http://ecology.bnu.edu.cn/zhangdy/TETRA/TETRA.htm

Liao, W. J., Zhu, B. R., Zeng, Y. F. and Zhang, D. Y. (2008) TETRA: an improved program for population genetic analysis of allotetraploid microsatellite data. *Molecular Ecology Resources* **8**, 1260-1262.

See Also

read.GeneMapper, write.Tetrasat, read.ATetra, read.GenoDive, read.Structure, dominant.to.codominant, read.SPAGeDi

Examples

```
## Not run:
# example from the Tetrasat website
mydata <- read.Tetrasat("http://markwith.freehomepage.com/sample.txt")
mydata$PopData
mydata$Genotypes["BCRHE3",]
mydata$Genotypes[["BR6","B1_Gtype"]]

## End(Not run)

# example with defined data:
cat("Sample Data", "A1_Gtype", "A10_Gtype", "B1_Gtype", "D7_Gtype",
"D9_Gtype", "D12_Gtype", "Pop",
"BCRHE 1              0406     04040404 0208     02020202 03030303 0710",
"BCRHE 10             0406     04040404 07070707 02020202 0304     0710",
"BCRHE 2              04040404 04040404 0708     02020202 010305   0710",
"BCRHE 3              04040404 04040404 02020202 0203     03030303 0809",
"BCRHE 4              04040404 04040404 0608     0203     03030303 070910",
"BCRHE 5              04040404 04040404 0208     02020202 03030303 050710",
"BCRHE 6              0304     04040404 0207     02020202 03030303 07070707",
"BCRHE 7              0406     04040404 0708     02020202 03030303 07070707",
"BCRHE 8              0304     04040404 0203     0203     03030303 0709",
"BCRHE 9              0406     04040404 0708     02020202 03030303 0710",
"Pop",
"BR 1                 0406     04040404 05050505 02020202 03030303 1012",
"BR 10                030406   04040404 0607     02020202 03030303 1011",
"BR 2                 030406   04040404 07070707 02020202 03030303 09090909",
"BR 3                 010304   04040404 07070707 02020202 03030303 09090909",
"BR 4                 030406   04040404 07070707 0203     03030303 10101010",
"BR 5                 030406   04040404 07070707 02020202 03030303 10101010",
"BR 6                 0406     04040404 0507     0203     03030303 10101010",
"BR 7                 0304     04040404 0809     02020202 03030303 070910",
"BR 8                 030406   04040404 07070707 02020202 03030303 070910",
"BR 9                 0406     04040404 07070707 02020202 03030303 07070707",
sep="\n", file="TetrasatExample.txt")
mydata2 <- read.Tetrasat("TetrasatExample.txt")
```

testgenotypes *Rubus Genotype Data for Learning polysat*

Description

Contains alleles of 20 Rubus samples at three microsatellite loci.

Usage

```
data(testgenotypes)
```

Format

A two-dimensional list of integer vectors. Samples are the first dimension of the list and loci the second dimension. Each vector contains all alleles for a given sample and locus.

Source

Clark and Jasieniuk, unpublished data

See Also

FCRinfo

write.ATetra *Write Genotypes in ATetra Format*

Description

Given a genotype object in the standard polysat format and an index of which samples belong to which populations, `write.ATetra` creates a text file of genotypes in the ATetra format.

Usage

```
write.ATetra(gendata, samples = dimnames(gendata)[[1]],
loci = dimnames(gendata)[[2]], popinfo = rep(1, length(samples)),
popnames = "onebigpop", commentline = "insert data info here",
missing = -9, file = "")
```

Arguments

`gendata`	A two-dimensional list of integer vectors, in the standard polysat format. Samples are represented in the first dimension and loci are represented in the second dimension, and both dimensions are named accordingly. Each vector contains each unique allele once.
`samples`	A character vector of samples to write to the file. This is a subset of `dimnames(gendata)[[1]]`.
`loci`	A character vector of loci to write to the file. This is a subset of `dimnames(gendata)[[2]]`.
`popinfo`	An integer vector where each number represents to which population a sample belongs. The vector should have names corresponding to `samples`, or if unnamed is assumed to be in the same order as `samples`.

popnames	A character vector containing the name of each population. The populations should be ordered by the numbers used to represent them in popinfo.
commentline	A character string to be put into the title line of the file.
missing	The symbol used to represent missing data in gendata.
file	A character string indicating the path and name to which to write the file.

Details

Note that missing data are not allowed in ATetra, although write.ATetra will still process missing data. When it does so, it leaves all alleles blank in the file for that particular sample and locus, and also prints a warning indicating which sample and locus had missing data.

popinfo can contain information for additional samples that are not to be used in the file (are not in samples). If the number of poplations in popinfo is greater than the length of popnames, a warning will be printed and only the populations up to the length of popnames will be used. If popnames is longer than the number of populations in popinfo, the same warning will be printed and there will be a subscripting error.

Value

A file is written but no value is returned.

Author(s)

Lindsay V. Clark

References

http://www.vub.ac.be/APNA/ATetra_Manual-1-1.pdf

van Puyvelde, K., van Geert, A. and Triest, L. (2010) ATETRA, a new software program to analyze tetraploid microsatellite data: comparison with TETRA and TETRASAT. *Molecular Ecology Resources* **10**, 331-334.

See Also

read.ATetra, write.Tetrasat, write.GenoDive, write.Structure, write.GeneMapper, codominant.to.dominant, write.SPAGeDi

Examples

```
# set up sample data (usually done by reading files)
mysamples <- c("ind1", "ind2", "ind3", "ind4")
myloci <- c("loc1", "loc2")
mygendata <- array(list(-9), dim=c(4,2), dimnames=list(mysamples, myloci))
mygendata[,"loc1"] <- list(c(202,204), c(204), c(200,206,208,212),
                           c(198,204,208))
mygendata[,"loc2"] <- list(c(78,81,84), c(75,90,93,96,99), c(87), c(-9))
mypopinfo <- c(1,2,1,2)
names(mypopinfo) <- mysamples
mypopnames <- c("this pop", "that pop")

# write an ATetra file
write.ATetra(mygendata, popinfo=mypopinfo, popnames=mypopnames,
             commentline="sample data", file="atetratest.txt")
```

```
# view the file
cat(readLines("atetratest.txt"),sep="\n")
```

`write.GeneMapper` *Write Genotypes to a Table Similarly to ABI GeneMapper*

Description

Given a genotype object in the standard polysat format, `write.GeneMapper` writes a text file of a table containing columns for sample name, locus, and alleles.

Usage

```
write.GeneMapper(gendata, file = "", samples = dimnames(gendata)[[1]],
loci = dimnames(gendata)[[2]])
```

Arguments

`gendata`	Genotypes in the standard polysat format: a two-dimensional list of vectors. Samples are represented in the first dimension of the list and loci in the second dimension of the list, and both dimensions are named accordingly. Each vector contains all unique alleles for a given sample and locus.
`file`	Character string. The path to which to write the file.
`samples`	Character vector. Samples to write to the file. This should be a subset of `dimnames(gendata)[[1]]`.
`loci`	Character vector. Loci to write to the file. This should be a subset of `dimnames(gendata)[[2]]`.

Details

Although I do not know of any population genetic software that will read this data format directly, the ABI GeneMapper Genotypes Table format is a convenient way for the user to store microsatellite genotype data and view it in a text editor or spreadsheet software. Each row contains the sample name, locus name, and alleles separated by tabs.

The number of allele columns needed is automatically detected by the function by using `estimate.ploidy` to find the maximum number of alleles per genotype.

`write.GeneMapper` handles missing data in a very simple way, in that it writes the missing data symbol directly to the table as though it were an allele. If you want missing data to be represented differently in the table, you can open it in spreadsheet software and use find/replace or conditional formatting to locate missing data.

The file that is produced can be read back into R directly by `read.GeneMapper`, and therfore may be a convenient way to back up genotype data for future analysis and manipulation in polysat. (`save` can also be used to back up an R object more directly.) If you need to edit the genotypes and would rather do it using spreadsheet software than in polysat (e.g. `mygendata[["ind1","loc3"]]<-c(122,126,134)`) writing the file in GeneMapper format, editing it, then reading it back in is an option.

Value

A file is written but no value is returned.

Author(s)

Lindsay V. Clark

References

http://www.appliedbiosystems.com/genemapper

See Also

read.GeneMapper, write.Structure, write.GenoDive, write.Tetrasat, write.ATetra, codominant.to.dominant, write.SPAGeDi

Examples

```
# create a genotype object (usually done by reading a file)
mysamples <- c("ind1", "ind2", "ind3", "ind4")
myloci <- c("loc1", "loc2")
mygendata <- array(list(-9), dim=c(4,2), dimnames=list(mysamples, myloci))
mygendata[,"loc1"] <- list(c(202,204), c(204), c(200,206,208,212),
                           c(198,204,208))
mygendata[,"loc2"] <- list(c(78,81,84), c(75,90,93,96,99), c(87), c(-9))

# write a GeneMapper file
write.GeneMapper(mygendata, "exampleGMoutput.txt")

# view the file with read.table
read.table("exampleGMoutput.txt", sep="\t", header=TRUE)
```

---

write.GenoDive *Write a File in GenoDive Format*

Description

write.GenoDive takes the standard genotype object used by polysat and creates a file formatted for the software GenoDive.

Usage

```
write.GenoDive(gendata, popnames = "onebigpop",
commentline = "file description goes here", digits = 2, file = "",
samples = dimnames(gendata)[[1]], loci = dimnames(gendata)[[2]],
popinfo = rep(1, times = length(samples)),
usatnts = rep(2, times = length(loci)), missing=-9)
```

Arguments

gendata	A two-dimensional list of vectors representing genotypes, in the standard polysat format. Samples are the first dimension of the list and loci are the second dimension. Dimensions are named accordingly. Each vector is numerical and contains all unique alleles for a given sample and locus.
popnames	A character vector of population names, ordered by the population numbers used in popinfo.

commentline	Character string. The first line of the file, containing a description of the data.
digits	Integer. How many digits to use to represent each allele (usually 2 or 3).
file	A character string of the file path to which to write.
samples	A character vector of samples to include in the file. A subset of dimnames(gendata)[[1]].
loci	A character vector of loci to include in the file. A subset of dimnames(gendata)[[2]].
popinfo	A numeric or integer vector containing the population number of each individual. names(popinfo) should include all of samples. If the vector is unnamed, it is assumed to be in the same order as samples.
usatnts	A numeric or integer vector, with one element for each locus. names(usatnts) should include all of loci, or if the vector is unnamed it is assumed to be in the same order as loci. Each element represents the length of the nucleotide repeat for the locus, such as 2 for dinucleotide repeats or 3 for trinucleotide repeats. If the alleles are already expressed in gendata as repeat number rather than fragment length in nucleotides, 1 should be the entry for that locus.
missing	The symbol used to indicate missing data in the input.

Details

The number of individuals, number of populations, number of loci, and maximum ploidy of the sample are calculated automatically and entered in the second line of the file. If the maximum ploidy needs to be reduced by random removal of alleles, it is possible to do this in the software GenoDive after importing the data.

Population names and population identies of individuals can optionally be entered as arguments in the file. If you do not already have this data readily available in R, you may prefer to enter it manually using a text editor or spreadsheet software after the file is written by write.GenoDive.

Several steps happen in order to convert alleles to the right format. First, all instances of the missing data symbol are replaced with zero. Alleles are then divided by the numbers provided in usatnts (and rounded down if necessary) in order to convert them from nucleotides to repeat numbers. If the alleles are still too big to be represented by the number of digits specified, write.GenoDive repeatedly subtracts a number (10^(digits-1), so 10 if digits=2) from all alleles at a locus until the alleles are small enough. Alleles are then converted to characters, and a leading zero is added to an allele if it does not have enough digits. These alleles are concatenated at each locus so that each sample*locus genotype is an uninterrupted string of numbers.

Value

A file is written but no value is returned.

Author(s)

Lindsay V. Clark

References

Meirmans, P. G. and Van Tienderen P. H. (2004) GENOTYPE and GENODIVE: two programs for the analysis of genetic diversity of asexual organisms *Molecular Ecology Notes* **4**, 792-794.

http://www.bentleydrummer.nl/software/software/GenoDive.html

See Also

read.GenoDive, write.Structure, write.ATetra, write.Tetrasat, write.GeneMapper, codominant.to.dominant, write.SPAGeDi

Examples

```
# set up the genotype object (usually done by reading a file)
mysamples <- c("Mal", "Inara", "Kaylee", "Simon", "River", "Zoe",
               "Wash", "Jayne", "Book")
myloci <- c("loc1", "loc2")
mygendata <- array(list(-9), dim=c(9,2), dimnames=list(mysamples,
                                                       myloci))
mygendata[,"loc1"] <- list(c(304,306), c(302,310), c(306), c(312,314),
c(312,314), c(308,310), c(312), c(302,308,310), c(-9))
mygendata[,"loc2"] <- list(c(118,133), c(121,130), c(122,139),
c(124,133), c(118,124), c(121,127), c(124,136), c(124,127,136),
c(121,130))

# make the vector of nucleotide repeat lengths
myusatnts <- c(2,3)
names(myusatnts) <- myloci

# set up population info (some or all of this might also be read from a file)
mypopnames <- c("Core","Outer Rim")
mypopinfo <- c(rep(1,4) ,rep(2,6))
names(mypopinfo) <- c("Simon", "River", "Inara", "Book", "Kaylee",
                      "Mal", "Wash", "Zoe", "Badger", "Jayne")

# write files (use file="" to display in console instead)
write.GenoDive(mygendata, usatnts=myusatnts, popnames=mypopnames,
  popinfo=mypopinfo, digits=2, commentline="Serenity crew",
  file="testGenoDive2.txt")
write.GenoDive(mygendata, usatnts=myusatnts, popnames=mypopnames,
  popinfo=mypopinfo, digits=3, commentline="Serenity crew",
  file="testGenoDive3.txt")
```

---

write.SPAGeDi *Write Genotypes in SPAGeDi Format*

Description

write.SPAGeDi takes a genotype object in the standard polysat format and creates a file that can be read by the software SPAGeDi. The user controls how the genotypes are formatted, and can specify the ploidy, population, and spatial coordinates of each sample.

Usage

```
write.SPAGeDi(gendata, samples = dimnames(gendata)[[1]],
              loci = dimnames(gendata)[[2]],
              indploidies = rep(4, length(samples)),
              popinfo = rep(1, length(samples)), allelesep = "/",
              digits = 2, file = "",
              spatcoord = data.frame(X = rep(1, length(samples)),
```

```
                                        Y = rep(1, length(samples)),
                                        row.names = samples),
                 usatnts = rep(2, length(loci)), missing = -9)
```

Arguments

gendata	A genotype object in the standard polysat format. A two dimensional list of integer vectors, where samples are represented and named in the first dimension, and loci are represented and named in the second dimension. Each vector contains all unique allele for a given sample and locus.
samples	Character vector. Samples to write to the file. Must be a subset of dimnames(gendata)[[1]].
loci	Character vector. Loci to write to the file. Must be a subset of dimnames(gendata)[[2]].
indploidies	Integer vector. Ploidy of each sample. This can either be named by sample, or be in the same order as samples.
popinfo	Vector. Population identity (or category) of each individual. This can either be named by sample, or be in the same order as samples.
allelesep	The character that will be used to separate alleles within a genotype. If each allele should instead be a fixed number of digits, with no characters to delimit alleles, set allelesep = "".
digits	Integer. The number of digits used to represent each allele.
file	Character string. The file path to write to.
spatcoord	Data frame. Spatial coordinates of each sample. Column names are used for column names in the file. Row names indicate sample, or if absent it is assumed that the rows are in the same order as samples.
usatnts	Integer vector. Repeat length of each locus (eg. 2 to indicate dinucleotide repeats, or 3 to indicate trinucleotide repeats). If the alleles in gendata are already in repeat lengths rather than nucleotides, the value should be 1. The vector should be named by loci, or else is assumed to be in the same order as loci.
missing	The symbol used in gendata to indicate missing data.

Details

popinfo, indploidies, spatcoord, and usatnts can have more samples or loci than are to be written to the file, as long as they are named.

The first line of the file contains the number of individuals, number of categories, number of spatial coordinates, number of loci, number of digits for coding alleles, and maximum ploidy, and is generated automatically from the data provided.

The function does not write distance intervals to the file, but instead writes 0 to the second line.

All alleles for a given locus are divided by the usatnts value for that locus, after all missing data symbols have been replaced with zeros. If necessary, a multiple of 10 is subtracted from all alleles at a locus in order to get the alleles down to the right number of digits.

If a genotype has fewer alleles than the indploidies value for that sample, zeros are added up to the ploidy. If the genotype has more alleles than the ploidy, a random subset of alleles is used and a warning is printed. If the genotype has only one allele (is fully heterozygous), then that allele is replicated to the ploidy of the individual. Genotypes are then concatenated into strings, delimited by allelesep. If allelesep="", leading zeros are first added to alleles as necessary to make them the right number of digits.

Value

A file is written but no value is returned.

Author(s)

Lindsay V. Clark

References

http://ebe.ulb.ac.be/ebe/Software_files/manual_SPAGeDi_1-3.pdf

Hardy, O. J. and Vekemans, X. (2002) SPAGeDi: a versatile computer program to analyse spatial genetic structure at the individual or population levels. *Molecular Ecology Notes* **2**, 618-620.

See Also

read.SPAGeDi, write.GenoDive, write.Structure, write.GeneMapper, write.ATetra, write.Tetrasat, codominant.to.dominant

Examples

```
# set up data to write (usually read from a file)
mygendata <- array(list(-9), dim=c(4,2),
                   dimnames=list(c("ind1","ind2","ind3","ind4"),
                                 c("loc1", "loc2")))
mygendata["ind1",] <- list(c(102,106,108),c(207,210))
mygendata["ind2",] <- list(c(104),c(204,210))
mygendata["ind3",] <- list(c(100,102,108),c(201,213))
mygendata["ind4",] <- list(c(102,112),c(-9))
myploidies <- c(3,2,2,2)
names(myploidies) <- c("ind1","ind2","ind3","ind4")
myusatnts <- c(2,3)
names(myusatnts) <- c("loc1","loc2")
myspatcoord <- data.frame(X=c(27,29,24,30), Y=c(44,41,45,46),
                          row.names=c("ind1","ind2","ind3","ind4"))

# write a file
write.SPAGeDi(mygendata, indploidies = myploidies, usatnts = myusatnts,
              spatcoord = myspatcoord, file="SpagOutExample.txt")
```

---

write.Structure *Write Genotypes in Structure 2.3 Format*

Description

Given genotypes in the form of a two-dimensional list of vectors, write.Structure produces a text file of the genotypes in a format readable by Structure 2.2 and higher. The user specifies the overall ploidy of the file as well as the ploidy of each sample.

Usage

```
write.Structure(gendata, ploidy, file="",
samples=dimnames(gendata)[[1]], loci=dimnames(gendata)[[2]],
indploidies=rep(ploidy,times=length(samples)),
extracols=NULL, missing=-9)
```

Arguments

gendata	Genotypes stored as a two-dimensional list of vectors, in the standard polysat format.
ploidy	PLOIDY for Structure, i.e. how many rows per individual to write.
file	A character string of the file path where the file should be written to.
samples	An optional character vector listing the names of samples to be written to the file.
loci	An optional character vector listing the names of the loci to be written to the file.
indploidies	An integer vector containing the ploidy of each sample. `names(indploidies)` should correspond to `samples`, or if the vector is unnamed it is assumed to be in the same order as `samples`.
extracols	An array, with the first dimension names corresponding to `samples`, of PopData, PopFlag, LocData, Phenotype, or other values to be included in the extra columns in the file.
missing	The number used to indicate missing data.

Details

Structure 2.2 and higher can process polyploid microsatellite data, although 2.3.3 or higher is recommended for its improvements on polyploid handling. The input format of Structure requires that each locus take up one column and that each individual take up as many rows as the parameter PLOIDY. Because of the multiple rows per sample, each sample name must be duplicated, as well as any population, location, or phenotype data. Partially heterozygous genotypes also must have one arbitrary allele duplicated up to the ploidy of the sample, and samples that have a lower ploidy than that used in the file (for mixed polyploid data sets) must have a missing data symbol inserted up to fill in the extra rows. Additionally, if some samples have more alleles than PLOIDY (if you are using a lower PLOIDY to save processing time, or if there are extra alleles from scoring errors), some alleles must be randomly removed from the data. `write.Structure` performs this duplication, insertion, and random deletion of data.

The argument `samples` contains all of the sample names to be written to the file, and is used to index `gendata`, `indploidies`, and `extracols`. These sample names will also be used as row names in the Structure file. Each sample name should only be in the vector sample once, because `write.Structure` will duplicate the sample names a number of times as dictated by `ploidy`. Likewise, `indploidies` and `extracols` only need to contain data for each sample once. If `samples` isn't specified by the user it will be extracted from `gendata`.

In writing genotypes to the file, `write.Structure` compares the number of alleles in the genotype, the ploidy of the sample as stored in `indploidies`, and the ploidy of the file as stored in `ploidy`, and does one of six things (for a given sample x and locus loc):

1) If indploidies[x] is greater than or equal to ploidy, and length(gendata[[x,loc]]) is equal to ploidy, the genotype data is used as is.

2) If indploidies[x] is greater than or equal to ploidy, and length(gendata[[x,loc]]) is less than ploidy, the first allele is duplicated as many times as necessary for there to be as many alleles as ploidy.

3) If indploidies[x] is greater than or equal to ploidy, and length(gendata[[x,loc]]) is greater than ploidy, a random sample of the alleles, without replacement, is used as the genotype.

4) If indploidies[x] is less than ploidy, and length(gendata[[x,loc]]) is equal to indploidies[x], the genotype data is used as is and missing data symbols are inserted in the extra rows.

5) If indploidies[x] is less than ploidy, and length(gendata[[x,loc]]) is less than indploidies[x], the first allele is duplicated as many times as necessary for there to be as many alleles as indploidies[x], and missing data symbols are inserted in the extra rows.

6) If indploidies[x] is less than ploidy, and length(gendata[[x,loc]]) is greater than indploidies[x], a random sample, without replacement, of indploidies[x] alleles is used, and missing data symbols are inserted in the extra rows. (Alleles are removed even though there is room for them in the file.)

Two of the header rows that are optional for Structure are written by `write.Structure`. These are 'Marker Names', containing the names of loci supplied in `gendata`, and 'Recessive Alleles', which contains the missing data symbol once for each locus. This indicates to the program that all alleles are codominant with copy number ambiguity.

The output file requires a few small modifications, done in a text editor or spreadsheet software, in order to be read by Structure. In the upper left corner the words "rowlabel" and "missing" should be deleted. Likewise the first and second rows for any non-locus columns should be deleted if the extracols argument was used. These should include the second dimension names used in `extracols`, and zeros, respectively.

Value

No value is returned, but instead a file is written at the path specified.

Note

If `extracols` is a character array, make sure none of the elements contain white space.

Author(s)

Lindsay V. Clark

References

http://pritch.bsd.uchicago.edu/structure_software/release_versions/v2.3.3/structure_doc.pdf

Hubisz, M. J., Falush, D., Stephens, M. and Pritchard, J. K. (2009) Inferring weak population structure with the assistance of sample group information. *Molecular Ecology Resources* **9**, 1322-1332.

Falush, D., Stephens, M. and Pritchard, J. K. (2007) Inferences of population structure using multilocus genotype data: dominant markers and null alleles. *Molecular Ecology Notes* **7**, 574-578.

See Also

`read.Structure`, `write.GeneMapper`, `write.GenoDive`, `write.ATetra`, `write.Tetrasat`, `codominant.to.dominant`, `write.SPAGeDi`

Examples

```
# input genotype data (this is usually done by reading a file)
mygendata <- array(list(c(100,102,106,108,114,118),c(102,110),
                        c(98,100,104,108,110,112,116),c(102,106,112,118),
                        c(104,108,110),c(-9),
                        c(204),c(206,208,210,212,220,224,226),
                        c(202,206,208,212,214,218),c(200,204,206,208,212),
                        c(-9),c(202,206)),
                   dim=c(6,2), dimnames=list(c("ind1","ind2","ind3",
                                              "ind4","ind5","ind6"),
                                             c("locus1","locus2")))
```

```
# create a vector of sample names to be used.  Note that this excludes
#  ind6.
# Also note that this could be obtained as names(mygendata[[1]]).
mysamples <- c("ind1","ind2","ind3","ind4","ind5")

# create a vector of the ploidy of each sample.
# Note that some of the above genotypes have more or fewer alleles than
# the ploidy of the sample.
myploidies <- c(6,6,6,4,4)
names(myploidies) <- mysamples

# Create an array containing data for additional columns to be written
# to the file.  You might also prefer to just read this and the ploidies
# in from a file.
myexcols <- array(data=c(1,2,1,2,1,1,1,0,0,0),dim=c(5,2),
                  dimnames=list(mysamples, c("PopData","PopFlag")))

# Write the Structure file, with six rows per individual.
# Since outfile="", the data will be written to the console instead of a file.
write.Structure(mygendata, 6, "", samples=mysamples, indploidies=myploidies,
                extracols=myexcols)
```

`write.Tetrasat` *Write Genotype Data in Tetrasat Format*

Description

Given a genotype object in the format used by polysat, `write.Tetrasat` creates a file that can be read by the software Tetrasat and Tetra.

Usage

```
write.Tetrasat(gendata, commentline = "insert data description here",
samples = dimnames(gendata)[[1]], loci = dimnames(gendata)[[2]],
popinfo = rep(1, length(samples)), usatnts = rep(2, length(loci)),
file = "", missing = -9)
```

Arguments

`gendata`	Genotype data in the format created and used by other polysat functions. A two dimensional list of numerical vectors, where samples are represented in the first dimension and loci are represented in the second dimension, and dimensions are named accordingly. Each vector contains each unique allele once.
`commentline`	A character string to be written as the first line of the file.
`samples`	A character vector of samples to write to the file. Should be a subset of `dimnames(gendata)[[1]]`
`loci`	A character vector of loci to write to the file. Should be a subset of `dimnames(gendata)[[2]]`.
`popinfo`	An integer vector indicating the population number of each sample. The vector should be named using sample names, or should be in the same order as `samples`.

usatnts	An integer vector indicating the length of the nucleotide repeats for each locus. The vector should be named using locus names, or should be in the same order as loci. The value indicating dinucleotide repeats is 2, trinucleotide repeats would be represented by 3, etc. Note that if the alleles in gendata are already in terms of repeat number rather than fragment length, the value should be 1.
file	A character string indicating the file to which to write.
missing	The symbol used to indicate missing data in gendata.

Details

Tetrasat files are space-delimited text files in which all alleles at a locus are concatenated into a string eight characters long. Population names or numbers are not used in the file, but samples are ordered by population, with the line "Pop" delimiting populations.

write.Tetrasat divides each allele by the length of the repeat and rounds down in order to convert alleles to repeat numbers. If necessary, it subtracts a multiple of 10 from all alleles at a locus to make all allele values less than 100, or puts a zero in front of the number if it only has one digit. If the individual is fully homozygous at a locus, the single allele is repeated four times. If any genotype has more than four alleles, write.Tetrasat picks a random sample of four alleles without replacement, and prints a warning. Missing data are represented by blank spaces.

Sample names should be a maximum of 20 characters long in order for the file to be read correctly by Tetrasat or Tetra.

Value

A file is written but no value is returned.

Author(s)

Lindsay V. Clark

References

http://markwith.freehomepage.com/tetrasat.html

Markwith, S. H., Stewart, D. J. and Dyer, J. L. (2006) TETRASAT: a program for the population analysis of allotetraploid microsatellite data. *Molecular Ecology Notes* **6**, 586-589.

http://ecology.bnu.edu.cn/zhangdy/TETRA/TETRA.htm

Liao, W. J., Zhu, B. R., Zeng, Y. F. and Zhang, D. Y. (2008) TETRA: an improved program for population genetic analysis of allotetraploid microsatellite data. *Molecular Ecology Resources* **8**, 1260-1262.

See Also

read.Tetrasat, write.Structure, write.GeneMapper, write.ATetra, write.GenoDive, codominant.to.dominant, write.SPAGeDi

Examples

```
# set up sample data (usually done by reading files)
mysamples <- c("ind1", "ind2", "ind3", "ind4")
myloci <- c("loc1", "loc2")
myusatnts <- c(2, 3)
names(myusatnts) <- myloci
```

```
mygendata <- array(list(-9), dim=c(4,2), dimnames=list(mysamples, myloci))
mygendata[,"loc1"] <- list(c(202,204), c(204), c(200,206,208,212),
                           c(198,204,208))
mygendata[,"loc2"] <- list(c(78,81,84), c(75,90,93,96,99), c(87), c(-9))
mypopinfo <- c(1,2,1,2)
names(mypopinfo) <- mysamples

# write a Tetrasat file
write.Tetrasat(mygendata, popinfo=mypopinfo, usatnts=myusatnts,
               commentline="sample data", file="tetrasattest.txt")

# view the file
cat(readLines("tetrasattest.txt"),sep="\n")
```

Index